PROCEDE DE COMPACTAGE D'UN PROGRAMME DE TYPE CODE OBJET INTERMEDIAIRE EXECUTABLE DANS UN SYSTEME EMBARQUE MUNI DE RESSOURCES DE TRAITEMENT DE DONNEES, SYSTEME COMPACTEUR ET SYSTEME EMBARQUE MULTI-APPLICATIONS CORRESPONDANTS

La présente invention est relative à un procédé de compactage d'un programme de type code objet intermédiaire, exécutable dans un système embarqué muni de ressources de traitement de données et au système compacteur correspondant.

Les systèmes embarqués munis de ressources de traitement de données actuels permettent de remplir des fonctions de plus en plus complexes et de plus en plus nombreuses, en raison de l'optimisation croissante de l'adéquation entre le matériel, constitutif de ces objets portatifs, et des logiciels, ou plus particulièrement des programmes ou applications implantés dans ces derniers, afin de leur conférer une ou plusieurs fonctionnalités spécifiques. La notion de système embarqué recouvre tout système informatique portable, tel qu'objet portatif, carte à microprocesseur ou analogue, distinct d'un micro-ordinateur classique.

C'est en particulier le cas des cartes à microprocesseur, encore appelées cartes à puce, telles que représentées en figure 1a, pour lesquelles on utilise un compilateur pour engendrer des instructions et un interpréteur permettant d'assurer l'exécution de ces instructions par le microprocesseur, ainsi que représenté en figure 1b. De manière classique, ainsi que représenté sur la figure 1a, une carte à microprocesseur 10 comprend un système d'entrée/sortie 12, relié au microprocesseur 14, une mémoire RAM 16, une mémoire non volatile 18, constituée par

une mémoire morte ROM 18b et une mémoire programmable 18a. L'ensemble de ces éléments est relié au microprocesseur 14 par une liaison par BUS. Un module 20 de chiffrement/déchiffrement de données peut, le cas échéant, être prévu.

L'implantation de l'ensemble des éléments logiciels d'applications, tels que porte-monnaie électronique, commerce électronique ou santé, dans la mémoire programmable non volatile, de l'interpréteur en mémoire programmable non volatile ou en mémoire morte et du système d'exploitation, en mémoire morte ROM, est représentée en figure 1c.

Le code objet intermédiaire est engendré par le compilateur à partir d'un programme source, le plus souvent écrit en langage de haut niveau, à partir des caractères ASCII. Le programme source et le code objet intermédiaire correspondant peuvent être exécutés par tous les microprocesseurs usuels, car l'interpréteur assure l'adaptation logicielle des instructions standard du code objet intermédiaire en instructions directement exécutables par le microprocesseur.

A titre d'exemple non limitatif, les fabricants de cartes à microprocesseur ont récemment développé des interpréteurs implantés dans la mémoire morte ROM. Ce type d'interpréteur lit de façon séquentielle un programme ou code objet intermédiaire, support d'une application par exemple, chargé par exemple dans la mémoire programmable de la carte à microprocesseur. Chaque instruction standard de ce code objet intermédiaire est interprétée par l'interpréteur, puis exécutée par le microprocesseur. En règle générale, les instructions standard du code objet intermédiaire permettent de traiter des fonctions évoluées telles que le traitement arithmétique et la manipulation

d'objets. La notion d'objet concerne les objets informatiques tels que des listes, tableaux de données ou analogues.

Toutefois, en raison notamment du caractère portatif de ces cartes à microprocesseur, l'encombrement et la taille de ces dernières sont limités. Il en est de même de la taille de la mémoire programmable de ces derniers, laquelle est, par construction, limitée à quelques kilo-octets. Une telle limitation structurelle ne permet pas la mise en œuvre de gros programmes d'application.

En outre, la tendance actuelle de la mise en œuvre de systèmes embarqués multi-applications trouve une limitation rédhibitoire à la multiplication du nombre d'applications installées sur un même système embarqué ou carte à microprocesseur, à un nombre excédant rarement trois applications.

La présente invention a pour objet de remédier à l'inconvénient précité par la mise en œuvre d'un procédé de compactage d'un programme de type code objet intermédiaire, utilisable dans un système embarqué de type carte à microprocesseur, afin de libérer de l'espace mémoire dans la mémoire programmable de ce système embarqué et permettre ainsi l'implantation d'au moins une application supplémentaire, après compactage de cette dernière.

Un autre objet de la présente invention est en outre la mise en œuvre d'un système de compactage de programmes de type code objet intermédiaire permettant l'implantation d'un programme de type code objet intermédiaire compacté dans un système embarqué multi-applications muni de ressources de traitement de données permettant l'exécution de programmes de type code objet intermédiaires compactés en l'absence de modification notable de la durée d'exécution, et en

transparence totale vis-à-vis du processus inhérent à chaque application non compactée.

Le procédé de compactage, objet de l'invention, d'un programme de type code objet intermédiaire consistant en une suite d'instructions standard, ce système embarqué étant doté d'une mémoire et d'un interpréteur de langage du programme de type code objet intermédiaire en instructions d'un code objet directement exécutables par un microprocesseur et ce programme étant normalement mémorisé dans la mémoire de ce système embarqué, est remarquable en ce que l'on recherche dans le programme de type code objet intermédiaire des séquences identiques d'instructions standard successives et l'on soumet les séquences identiques d'instructions standard successives à un test de comparaison de supériorité d'une fonction d'au moins le nombre d'occurrences de ces séquences dans le programme de type code objet intermédiaire à une valeur de référence. Sur réponse positive au test précité, pour chaque séquence identique d'instructions standard successives satisfaisant à l'étape de test, on engendre une instruction spécifique par définition d'un code opératoire spécifique et association à ce code opératoire spécifique de la séquence d'instructions standard successives ayant satisfait à ce test. On remplace en outre, dans le programme de type code objet intermédiaire mémorisé, chaque occurrence de chaque séquence d'instructions standard successives par le code opératoire spécifique qui lui est associé, pour obtenir un programme de type code objet intermédiaire compacté, succession d'instructions standard et de codes opératoires spécifiques. On mémorise dans la mémoire une table de décompactage permettant la mise en correspondance biunivoque entre chaque code opératoire spécifique introduit et la séquence

d'instructions standard successives associée à ce dernier. Ce processus permet d'optimiser l'espace mémoire occupé par le programme de type code objet intermédiaire compacté par mémorisation dans la mémoire programmable d'une seule occurrence des séquences identiques d'instructions standard successives.

Le procédé, le système de compactage d'un programme de type code objet intermédiaire et le système embarqué multi-applications correspondant, objets de la présente invention, trouvent application dans le domaine technique des systèmes embarqués, plus particulièrement dans la mise en œuvre et la gestion de cartes à microprocesseur.

Ils seront mieux compris à la lecture de la description et à l'observation des dessins ci-après dans lesquels, outre les figures 1a à 1c relatives à l'art antérieur,

- la figure 2a représente un organigramme général illustratif d'un procédé de compactage d'un programme de type code objet intermédiaire, selon la présente invention ;

- la figure 2b représente un schéma synoptique illustratif de la mise en œuvre des différents opérateurs nécessaires à l'obtention d'un programme de type code objet intermédiaire compacté et de paramètres permettant le décompactage ou l'exécution de ce programme ;

- la figure 2c représente, à titre purement illustratif, l'implantation en mémoire programmable, non volatile, d'une carte à microprocesseur de ce programme de type code objet intermédiaire compacté et des paramètres d'exécution ou décompactage de ce dernier ;

- la figure 3a représente, dans un mode de réalisation particulier non limitatif, un schéma illustratif de la structure d'un premier fichier constitutif de ces

paramètres d'exécution ou décompactage de ce programme de type code objet intermédiaire compacté ;

- la figure 3b représente, dans un mode de réalisation particulier non limitatif, un schéma illustratif de la structure d'un deuxième fichier constitutif de ces paramètres d'exécution ou décompactage de ce programme de type code objet intermédiaire compacté ;

- la figure 4 représente, à titre illustratif, l'implantation en mémoire programmable non volatile d'un programme de type code objet intermédiaire compacté, selon la présente invention, dans une carte à microprocesseur ou système embarqué multi-applications ;

- la figure 5 représente, à titre illustratif, un processus de mise en œuvre spécifique du procédé de compactage d'un programme de type code objet intermédiaire dans lequel une actualisation des codes spécifiques relatifs à des applications ou programmes de type code objet intermédiaires distincts est réalisé ;

- les figures 6a et 6b représentent, sous forme d'éléments fonctionnels, un système de compactage d'un programme de type code objet intermédiaire conforme à l'objet de la présente invention.

Le procédé de compactage d'un programme de type code objet intermédiaire, conforme à l'objet de la présente invention, sera maintenant décrit en liaison avec la figure 2a. La désignation programme type code objet intermédiaire recouvre tout programme intermédiaire dans la présente demande de brevet.

Ce procédé sera décrit de manière non limitative dans le cas de sa mise en œuvre dans un système embarqué constitué par exemple par une carte à microprocesseur telle que représentée en figure 1a, ce programme de type code

objet intermédiaire étant obtenu de manière classique, ainsi que représenté en figure 1b, et l'implantation en mémoire programmable d'une pluralité d'applications de l'interpréteur et du système d'exploitation OS en mémoire ROM étant représentée en figure 1c, de manière non limitative.

Le programme de type code objet intermédiaire consiste en une suite d'instructions standard exécutables par le microprocesseur par l'intermédiaire de l'interpréteur.

Le procédé de compactage d'un tel programme consiste, préalablement à l'implantation de ce dernier en mémoire programmable 18a, à effectuer, ainsi que représenté en figure 2a, en une étape 1000, une recherche dans le programme de type code objet intermédiaire des séquences identiques d'instructions standard successives, ces séquences identiques étant notées Si. Par séquences identiques, on indique une suite d'un nombre n d'octets déterminé susceptible d'apparaître de manière répétitive dans le programme de type code objet intermédiaire précité. Ainsi, le rang i des séquences identiques indique, pour des valeurs de i différentes, des séquences distinctes. En outre, l'étape 1000 de recherche précitée consiste à déterminer le nombre d'occurrences N_i de chaque séquence identique S_i précitée. A l'issue de l'étape 1000 de recherche, on dispose d'une pluralité de séquences identiques S_i, chaque séquence S_i étant distincte, et d'un nombre N_i représentant le nombre d'occurrences dans le programme de type code objet intermédiaire de chacune des séquences S_i.

Suite à l'étape 1000 précitée, le procédé de compactage, objet de la présente invention, consiste à

soumettre, en une étape 1001, les séquences identiques d'instructions standard successives S_i à un test de comparaison d'une fonction $f(N_i)$ d'au moins le nombre d'occurrences N_i associé à une séquence identique S_i. Sur la figure 2a, le test de comparaison est noté :

$$f(N_i) > Ref.$$

Lorsque la réponse au test 1001 est négative, la fonction d'au moins le nombre d'occurrences N_i n'étant pas supérieure à la valeur de référence, le test 1001 est appliqué à la séquence identique suivante, de rang i+1, par incrémentation de l'indice i à l'étape 1002.

Les étapes 1000, 1001 et 1002 représentées en figure 2a permettent ainsi de rechercher dans le programme de type code objet intermédiaire l'ensemble des séquences ou séries d'octets identiques ou, à tout le moins, un nombre significatif donné de ces séquences identiques, ainsi qu'il sera décrit ultérieurement dans la description.

Sur réponse positive au test 1001 précité, le procédé de compactage, objet de la présente invention, consiste ensuite à engendrer une instruction spécifique, notée IS_i, par définition d'un code opératoire spécifique, noté C_i, et association à ce code opératoire spécifique de la séquence d'instructions standard successives ayant satisfait au test, la séquence d'instructions standard successives S_i. Sur la figure 2a, l'étape de création d'instructions spécifiques est notée :

$$IS_i = C_i : S_i.$$

On indique que l'étape de définition d'un code opératoire spécifique et d'association à ce dernier de la séquence d'instructions standard successives S_i peut consister en l'attribution d'une valeur de code et l'association sous

forme d'une liste par exemple de cette valeur de code et de la séquence d'instructions S_i précitée.

Suite à l'étape 1003, le procédé de compactage consiste ensuite, à l'étape 1004, à remplacer dans le programme de type code objet intermédiaire mémorisé chaque occurrence de la séquence d'instructions successives standard S_i par le code opératoire spécifique C_i qui lui est associé pour obtenir un programme de type code objet compacté, noté FCC, succession d'instructions standard et de codes opératoires spécifiques C_i.

Le processus de remplacement peut alors être réitéré pour chaque séquence ou série d'instructions standard identiques S_i tant que l'indice i est inférieur à un nombre P de séquences identiques, un test 1005 de comparaison de l'indice i à la valeur P permettant, sur réponse positive à ce test, le retour à l'étape d'incrémentation 1002 de l'indice i précédemment décrit.

On comprend en particulier que, suite à l'itération du processus de remplacement ainsi formé, on obtient un programme de type code objet compacté, noté FCC, auquel est associé un fichier d'exécution de ce dernier, fichier noté FEX, ce fichier d'exécution consistant au moins en une mise en correspondance biunivoque entre chaque code spécifique C_i et la séquence d'instructions standard successives S_i précitée.

Suite à l'obtention des deux fichiers précités, programme de type code objet intermédiaire compacté et fichier d'exécution, sur réponse négative au test 1005 par exemple, il est possible de procéder à une mémorisation, dans la mémoire programmable 18a par exemple, du programme de type code objet intermédiaire compacté obtenu FCC précité, et bien entendu du fichier d'exécution FEX

précédemment mentionné. La mémorisation précitée peut de manière non limitative être effectuée dans la mémoire non volatile 18, mémoire programmable 18a ou même mémoire morte 18b.

En ce qui concerne le test de comparaison 1001 précité, on indique bien entendu que la fonction d'au moins le nombre d'occurrences de chaque séquence identique S_i peut être définie de façon à obtenir une optimisation du gain de compactage ainsi réalisé. Dans un mode de réalisation non limitatif, on indique que cette fonction peut être établie de façon à réaliser une comparaison de la taille de chaque séquence identique d'instructions standard successives en nombre d'octets à une valeur de seuil, exprimée par exemple en nombre d'instructions standard.

La figure 2b décrit, à titre d'exemple illustratif, un mode opératoire permettant d'engendrer un programme de type code objet intermédiaire compacté conformément à la mise en œuvre du procédé, objet de la présente invention.

Dans un premier temps, le créateur du programme de type code objet intermédiaire réalise un fichier de type texte contenant le programme source. Ce programme établi par ce dernier à partir d'un langage évolué est, de manière générale, écrit en code ASCII de manière à être lu facilement et à pouvoir contenir des commentaires qui facilitent, d'une part, la compréhension, et d'autre part, la mise au point de ce dernier. Le programme source ainsi obtenu est introduit dans un compilateur de type classique, dit compilateur standard, dont le rôle consiste à transformer chaque ligne de programme en instructions exécutables ou, à tout le moins, en instructions interprétables pour obtenir un programme de type code objet

intermédiaire consistant en une suite d'instructions standard interprétables par l'interpréteur.

Le fichier de type code objet intermédiaire ainsi obtenu suite au processus de compilation est introduit dans un système compacteur permettant la mise en œuvre du procédé de compactage précédemment décrit en liaison avec la figure 1. Ce système compacteur sera décrit ultérieurement dans la description.

Le processus de compactage mis en œuvre, ainsi que décrit précédemment, permet alors l'obtention d'un fichier d'instructions interprétables FCC, c'est-à-dire du fichier constitutif du programme de type code objet intermédiaire compacté, et du fichier d'exécution FEX précédemment mentionné dans la description.

Le mode opératoire du système de compactage sera décrit ci-après dans un exemple spécifique de mise en œuvre.

En premier lieu, le système compacteur analyse toutes les instructions standard I_s et dresse une liste de toutes les séries d'instructions standard existant dans le fichier constitutif de ce dernier.

Si le fichier précité contient 1000 octets par exemple, le système compacteur lance une procédure de recherche de toutes les séries d'au moins deux octets jusqu'à un nombre Q par exemple. La recherche précitée peut être effectuée pour des séries de deux octets, puis de trois octets, et ainsi de suite jusqu'à Q octets. Dans un mode de réalisation préférentiel, le nombre Q avait la valeur 500.

Ainsi, pour chaque séquence d'instructions S_i, formée par une série d'instructions standard I_s, le système compacteur détermine si cette séquence S_i est déjà dans la liste. Dans un tel cas, le système compacteur rajoute une unité au nombre d'occurrences N_i de la séquence S_i précitée.

A la fin du processus de recherche précité, le système compacteur a ainsi engendré une liste complexe contenant l'ensemble des séquences d'instructions S_i examinées, à chaque séquence étant associé un nombre d'occurrences N_i dans le programme de type code objet intermédiaire considéré.

Un tableau illustratif est donné ci-après pour un programme de type code objet intermédiaire constitué par la série d'instructions ci-après :

1-7-3-5-7-3-7-3-5-7.

Alors que pour l'exemple illustratif donné dans le tableau, TABLEAU 1 ci-après, la série d'instructions précitée comporte dix instructions, chaque instruction étant représentée par un octet et illustrée par un chiffre de 1 à 7, les séquences d'instructions successives examinées comprennent 2, 3, 4 puis 5 octets.

Les séquences d'instructions successives S_i, dont le nombre d'occurrences dans le programme de type code objet intermédiaire précité est supérieur ou égal à deux, sont données dans le tableau ci-après.

TABLEAU 1

4 octets	[7-3-5-7]:2		
3 octets	[7-3-5]:2	[3-5-7]:2	
2 octets	[7-3]:3	[3-5]:2	[5-7]:2

En deuxième lieu, le système compacteur remplace certaines séquences S_i du TABLEAU 1 par un code d'instructions spécifiques.

Le code d'instructions spécifiques C_i est déterminé chronologiquement à partir du premier code correspondant à

une instruction standard. Dans un code intermédiaire objet courant, il existe à ce jour 106 instructions standard et les codes de ces instructions sont compris entre 000 et 105. Le premier code d'instructions spécifiques C_i peut alors être la valeur 106, le second la valeur 107 et ainsi de suite. Chaque fois que les séquences d'instructions identiques S_i sont remplacées par un nouveau code d'instructions spécifiques C_i, une fois qu'une telle opération est terminée, la liste représentée dans le tableau précédent est alors recalculée.

A titre d'exemple non limitatif et dans le cas du remplacement de la séquence d'instructions de 4 octets représentée au tableau précédent, la séquence 7-3-5-7, et allocation d'un code spécifique correspondant 106, le programme de type code objet intermédiaire compacté devient :

1-106-3-106.

Dans ces conditions, il n'existe plus de séquence d'instructions standard I_s et d'instructions spécifiques IS se retrouvant à l'identique au moins deux fois. Bien entendu, le fichier constitutif du programme de type code objet intermédiaire compressé FCC et le fichier d'exécution ou de décompactage de ce dernier sont mémorisés au niveau du système compacteur précité.

Après l'opération de compactage réalisée par le système compacteur, on dispose du programme de type code objet intermédiaire proprement dit, exécutable par le système cible, et du fichier d'exécution FEX précité. Le premier précité contient des instructions standard I_s et des instructions spécifiques IS, alors que le second comporte au moins un tableau permettant de lier les codes spécifiques C_i avec les séries d'instructions standard S_i remplacées par

les codes spécifiques précités. Bien entendu, ces deux fichiers peuvent être regroupés en un seul et même fichier en vue du transfert de ce dernier au système cible destinataire, c'est-à-dire à la carte à microprocesseur destinée à recevoir ce dernier.

En ce qui concerne le fichier d'exécution FEX, on indique que celui-ci comporte au moins un fichier, noté MEM-SEQ, constitué par une succession de plusieurs champs tels qu'un champ de code spécifique C_i, un champ de séquence S_i, tel que mentionné précédemment.

Suite à l'opération précitée, le fichier unique ou, le cas échéant, les deux fichiers précités, sont transmis au système cible et directement traités par un programme de chargement. Ce programme de chargement est principalement chargé d'écrire en mémoire programmable 18a ou en mémoire morte 18b les données reçues en vue d'une bonne exécution par la suite.

A titre d'exemple non limitatif, on indique que le fichier relatif au programme de type code objet intermédiaire compacté FCC est stocké sans traitement à partir d'une adresse déterminée, notée ADR-MEM-PGM, dans la mémoire programmable 18a précitée.

En ce qui concerne le fichier d'exécution FEX, on indique que vis-à-vis de ce dernier, le programme de chargement analyse les données de ce fichier et créé dynamiquement un tableau, noté TAB-PRO, permettant d'associer les codes d'instructions spécifiques C_i avec les séries d'instructions. En fait, le tableau TAB-PRO permet d'assurer une correspondance biunivoque entre les codes d'instructions spécifiques précités C_i et une adresse d'implantation permettant l'exécution des instructions correspondantes.

Une implantation, d'une part, du fichier support du programme de type code objet intermédiaire compacté FCC, du fichier d'exécution FEX et du fichier TAB-PRO précédemment cité, ce dernier fichier ayant été engendré par le programme de chargement dans la mémoire programmable 18a de la carte à microprocesseur, est représentée en figure 2c.

Sur cette figure, alors que le tableau des codes d'instructions standard I_s est mémorisé au niveau de l'interpréteur en un tableau TAB-STD, le fichier d'exécution FEX et le fichier TAB-PRO permettant d'assurer la correspondance des sauts d'adresse avec les codes d'instructions spécifiques C_i, ces deux tableaux permettant l'exécution effective au niveau du microprocesseur de l'unité cible du programme de type code objet intermédiaire compacté FCC, sont au contraire mémorisés dans la mémoire programmable 18a. On dispose ainsi d'un ensemble exécutable par l'intermédiaire de l'interpréteur dans les conditions qui seront décrites ci-après.

Préalablement à la description de l'exécution d'un programme de type code objet intermédiaire compacté FCC, une description détaillée de la structure des fichiers d'exécution FEX et du fichier TAB-PRO et de la relation fonctionnelle entre ces derniers sera maintenant donnée en liaison avec les figures 3a et 3b.

Sur la figure 3a, on a représenté le fichier d'exécution FEX de manière détaillée, celui-ci comportant ainsi que mentionné précédemment, outre les champs de codes spécifiques C_i et de séquences d'instructions S_i, un champ de fin de macro-instructions, noté FM, indiquant en fait la fin de la séquence précitée. Dans un mode de réalisation non limitatif, chaque code spécifique C_i peut être inscrit au début du champ, sur un octet par exemple, puis chaque

séquence correspondante S_i est inscrite dans un second champ de longueur variable. Le code de fin de macro FM est de type standard et correspond à celui utilisé par le langage classique précédemment indiqué dans la description.

Lors de la réception du fichier d'exécution FEX dont la structure de données correspond à celle représentée en figure 3a par exemple, les différents champs C_i, S_i et FM sont traités séparément.

En premier lieu, le code spécifique C_i de l'instruction spécifique IS correspondante est écrit dans le fichier TAB-PRO et la séquence d'instructions S_i associée à ce code spécifique constitutive de l'instruction spécifique précitée est écrite dans un fichier ou mémoire référencée MEM-SEQ à partir d'une adresse notée ADR-1. Le code C_i de l'instruction spécifique correspondante est écrit à l'adresse TAB-PRO + 3 x (CODE-106). Dans cette relation, on indique que l'adresse TAB-PRO est l'adresse d'ouverture du fichier TAB-PRO, alors que la valeur CODE représente la valeur numérique du code C_i correspondant. Sur la figure 3b, on a représenté le mode opératoire correspondant pour une valeur d'adresse TAB-PRO égale arbitrairement à 0, le premier code spécifique alloué ayant la valeur 106 et les autres codes spécifiques alloués successifs ayant des valeurs 107 et suivantes. On indique que sur la figure 3b seuls quatre codes spécifiques 106, 107, 110 et 120 ont été représentés pour une meilleure compréhension, les autres espaces mémoire étant remplis par des valeurs arbitraires.

Dans ces conditions, Adr-i est la première adresse disponible dans la mémoire MEM-SEQ, cette adresse correspondant à l'adresse Adr-1 pour la première séquence d'instructions $S_i = S_1$. A partir de cette première adresse, laquelle constitue l'adresse d'ouverture du fichier dans la

mémoire MEM-SEQ, les séquences d'instructions S_i sont ainsi écrites de façon séquentielle dans l'ordre de leur chargement. Le code FM de fin de macro est également écrit à la fin de la série correspondante.

A la suite de l'écriture précitée dans la mémoire MEM-SEQ et après une étape de vérification correcte d'écriture, le programme de chargement écrit dans le tableau TAB-PRO à la suite de chaque code spécifique C_i la valeur de l'adresse d'écriture de la séquence dans la mémoire MEM-SEQ. Le programme de chargement recalcule alors une nouvelle adresse d'écriture pour la prochaine séquence S_i de rang i incrémenté ou décrémenté en fonction du mode de parcours des séquences d'instructions S_i précitées.

Un processus d'exécution d'un programme de type code objet intermédiaire compacté supporté par un fichier FCC précédemment décrit et contenant des instructions spécifiques sera maintenant décrit en référence à la figure 4.

L'exécution d'un tel programme s'effectue par l'intermédiaire de l'interpréteur à l'aide d'un pointeur d'instruction, noté PI. En fait, le pointeur d'instruction PI lit le code de l'instruction à exécuter, instruction standard Is ou instruction spécifique IS, et présente ce code à l'interpréteur qui déclenche ensuite les actions correspondant à ce dernier.

Au début de l'exécution d'un programme, le pointeur d'instruction PI est chargé avec l'adresse de début de ce programme, c'est-à-dire l'adresse ADR-MEM-PGM.

L'interpréteur analyse la valeur du code lu par le pointeur d'instruction PI. Dans le cadre de cette analyse, ce dernier détermine si cette valeur de code correspond à un code de type standard Cs ou au contraire à un code de type

spécifique C_i. Cette opération est réalisée à partir du tableau TAB-STB mémorisé au niveau de l'interpréteur et associant les codes d'instructions standard, et donc les instructions standard Is, avec les adresses d'exécution dans son programme.

Si la valeur du code lu n'est pas dans ce dernier tableau, l'interpréteur provoque un appel en lecture dans le tableau TAB-PRO afin de vérifier l'existence de la valeur du code lu dans ce dernier tableau. Si le code lu n'est pas non plus dans ce dernier tableau, l'interpréteur est incapable d'exécuter l'instruction lue et l'exécution du programme s'arrête en indiquant un message d'erreur, non décrit dans l'organigramme de la figure 4.

Sur la figure 4 précitée, on a représenté par 2000 le début de l'opération d'exécution, 2001 l'opération d'initialisation du pointeur d'instruction PI à la première instruction du programme et 2002 une opération de lecture de l'instruction pointée par le pointeur d'instruction PI. Cette opération correspond en fait à la lecture de la valeur de code précitée.

De la même manière, à l'étape 2003 de la figure 4, l'appartenance ou la non-appartenance de la valeur de code lu au tableau des codes standard TAB-STB et l'appartenance de cette valeur de code lu au tableau TAB-PRO permet en fait de constituer le test 2003 précité, l'instruction lue INS étant ainsi discriminée en qualité d'instruction standard Is ou instruction spécifique IS. La situation d'absence d'appartenance du code lu et de l'instruction lue correspondante à l'un et l'autre des deux tableaux génératrice d'un message d'erreur n'est pas représentée en figure 4, afin de ne pas surcharger le dessin.

Si, sur réponse positive au test 2003 précité, le code lu correspond à une instruction spécifique, la valeur du pointeur d'instruction PI, pour lire l'instruction suivante, est calculée et mémorisée dans la pile. L'interpréteur lit dans le tableau TAB-PRO la valeur de l'adresse de la séquence d'instructions Si associée au code spécifique C_i lu et initialise la valeur du pointeur d'instruction PI avec cette valeur. L'ensemble de ces opérations porte la référence 2004 sur la figure 4 précitée. A la suite de l'étape 2004 précitée, l'interpréteur boucle de nouveau à l'étape lecture du code, ainsi que représenté en figure 4, par retour à l'étape 2002.

Si, sur réponse négative au test 2003, le code lu correspond à une instruction de type standard Is, l'interpréteur contrôle dans une étape de test 2005 si la valeur de ce code correspond à une valeur de fin de macro représentant en fait une fin de séquence. Si tel est le cas, la valeur précédemment mémorisée dans la mémoire de pile est extraite et la pile est mise à jour, cette valeur étant chargée dans le pointeur d'instruction PI. L'opération d'extraction de la pile de la valeur précédemment mémorisée constituant une adresse de retour puis de remise à jour de la pile, est représentée en 2006, l'adresse de retour étant notée ADR-RET. Suite à l'étape 2006 précitée, l'interpréteur boucle de nouveau le processus à l'étape de lecture de la valeur de code, c'est-à-dire à l'étape 2002. Si, sur réponse négative au test 2005, la valeur du code lu correspondant à une instruction de type standard ne correspond toutefois pas à une fin de macro ou fin de série, alors, le code est exécuté de façon connue en tant que telle par l'interpréteur. Ainsi qu'on l'a toutefois représenté en figure 4, une étape de test 2007 est prévue dans ce cas

préalablement à l'exécution proprement dite de l'instruction standard précitée. Le test 2007 consiste à vérifier que la valeur du code et l'instruction INS correspondante ne correspond pas à celle d'une fin de programme. Sur réponse positive au test 2007 précité, l'étape d'exécution 2008 de cette instruction par l'interpréteur est alors réalisée, à cette étape d'exécution étant associée une étape d'incrémentation du pointeur d'instruction vers l'instruction suivante. Suite à l'étape 2008 précitée, l'interpréteur reboucle vers l'étape de lecture de la valeur de code pointée par le pointeur d'instruction PI, c'est-à-dire l'étape de lecture 2002.

Sur réponse négative au test 2007, l'instruction correspondant à une instruction de fin de programme, une étape de fin 2009 est réalisée. L'interpréteur dans ce cas arrête son action et donne la main au système d'exploitation OS. Celui-ci attend alors une nouvelle instruction de commande.

Le mode de réalisation et de mise en œuvre du processus d'exécution d'un programme de type code objet intermédiaire compacté, tel que décrit précédemment en liaison avec la figure 4, n'est pas limitatif.

En premier lieu, on indique que la mémoire de pile peut être subdivisée en deux mémoires de pile séparées, une mémoire de pile pour les instructions standard Is, et une mémoire de pile pour les instructions spécifiques IS encore désignées par macro-instructions. Dans un tel mode de réalisation, on connaît le nombre maximal d'imbrications d'instructions spécifiques IS intraprocéduralement. Pour avoir la taille totale occupée par cette pile, il suffit de multiplier par le nombre maximal de procédures imbriquées. La mise en œuvre d'une mémoire de pile séparée pour les

instructions spécifiques IS procure, par rapport à l'utilisation d'une seule pile, une réduction de la consommation totale de mémoire.

En outre, afin d'augmenter le nombre d'instructions spécifiques IS utilisables en lieu et place du nombre d'instructions spécifiques limité entre 106 et 255 dans l'exemple précédemment donné dans la description, les codes spécifiques C_i peuvent avantageusement être codés sur deux octets. Dans ces conditions, une valeur de code particulière, telle que la valeur 255, peut alors indiquer le codage sur deux octets.

Enfin, le système cible, lorsque ce dernier est constitué par un système embarqué multi-applications, comprend plusieurs programmes compilés et compactés, c'est-à-dire plusieurs fichiers FCC précédemment décrits dans la description. Ces programmes doivent fonctionner de manière indépendante. Dans un tel cas, l'interpréteur étant unique, il exécute tous les programmes d'applications chargés par le programme de chargement. Si deux programmes d'applications utilisent des instructions spécifiques, dans le mode de réalisation précédemment décrit dans la description, il est possible que le système compacteur affecte le même code spécifique C_i pour deux séries d'instructions différentes.

Afin de remédier à une telle situation et pour permettre à l'interpréteur de distinguer les deux codes, les champs du fichier d'exécution FEX tels que représentés précédemment en figure 3a peuvent être complétés par un troisième paramètre relatif à un numéro d'identification de l'application considérée. Ce numéro d'identification est alors mémorisé également pour chaque code spécifique affecté dans le tableau TAB-PRO. Ce dernier paramètre constitue en fait la référence du programme chargé en même temps que le

fichier contenant le tableau permettant d'associer chaque code d'instruction spécifique C_i avec des séquences d'instructions S_i remplacées par ces derniers pour l'application considérée. Lors de l'exécution de l'application du programme par l'interpréteur, ce dernier peut ainsi assurer la discrimination des instructions spécifiques relatives à cette application.

Bien entendu, le processus précédemment décrit permettant la mise en œuvre d'un système embarqué multi-applications présente l'inconvénient d'une consommation accrue de mémoire, du fait de l'attribution d'un champ supplémentaire relatif au numéro d'application considérée.

Un processus plus avantageux sera maintenant décrit en liaison avec la figure 5.

Relativement à la figure 5, on considère un système embarqué tel qu'une carte à microprocesseur comportant plusieurs applications, notées A_1 à A_k, les valeurs A_1 à A_k constituant en fait des numéros d'identification de chaque application. Dans ce but, lors du compactage, conformément au procédé objet de la présente invention tel que décrit précédemment dans la description, de tout programme ou application source de numéro d'identification donné à A_1 à A_{k-1} par exemple, le système cible, c'est-à-dire carte à microprocesseur, transmet au compacteur le contenu de la mémoire MEM-SEQ avec bien entendu les codes spécifiques C_i correspondants. En fait, le système cible recalcule à partir du fichier ou tableau TAB-PRO et du contenu de la mémoire MEM-SEQ un fichier des coefficients spécifiques antérieurs, noté F-C-ANT, relatif aux applications A_1 à A_{k-1}. Le fichier F-C-ANT assure la mise en correspondance biunivoque de chaque code spécifique C_i et de la séquence Si associée à ce dernier pour l'ensemble des applications A_1 à A_{k-1}. Dans ces

conditions et dans un mode de réalisation non limitatif simplifié, le fichier F-C-ANT peut consister en un fichier de même format que le fichier FEX précité. Dans le processus de compactage préférentiel tel que représenté en figure 5, le fichier F-C-ANT des codes spécifiques antérieurs est alors communiqué au compacteur afin d'assurer un apprentissage de ce dernier.

Lors du compactage d'une nouvelle application, de numéro d'identification A_k, le compacteur recherche toutes les occurrences des séquences d'instructions S_i déjà enregistrées dans le fichier F-C-ANT, c'est-à-dire en fait dans le tableau TAB-PRO du système cible pour les applications antérieures A_1 à A_{k-1}. A chaque occurrence trouvée, le système compacteur remplace la séquence d'instructions correspondante S_i par le code spécifique C_i de l'instruction spécifique IS correspondante. Cette opération étant effectuée, le système compacteur peut alors analyser l'application de code d'identification A_k et bien entendu rechercher d'autres occurrences en vue de créer des instructions spécifiques supplémentaires qui n'ont pas encore été mémorisées. Une mise à jour du fichier F-C-ANT peut alors être effectuée. Le processus de décompactage décrit en liaison avec la figure 5 peut être mis en œuvre de manière particulièrement avantageuse pour assurer le compactage, soit de programmes chargés pour la première fois dans le système embarqué, soit de programmes chargés en supplément à d'autres programmes compactés existants dans le système embarqué.

Dans les deux hypothèses précitées, le procédé de compactage, objet de l'invention, consiste à mémoriser la table d'exécution relative à au moins un programme intermédiaire de type code objet compacté, le premier de ces

programmes dans la première hypothèse et un ou plusieurs programmes compactés existants dans la deuxième hypothèse, puis pour tout programme intermédiaire supplémentaire, à lire la table d'exécution mémorisée et à effectuer le compactage de tout programme supplémentaire, en tenant compte des instructions et codes spécifiques mémorisés dans la table d'exécution, ainsi que décrit précédemment dans la description. Bien entendu, le programme de type code objet intermédiaire compacté ainsi créé ne peut alors être exécuté que sur le système cible qui a fourni précédemment au système compacteur le fichier F-C-ANT pertinent correspondant.

Dans le cadre de la mise en œuvre du procédé de compactage d'un programme de type code objet intermédiaire, tout système embarqué, tel qu'un objet portatif multi-applications formé par exemple par une carte à microprocesseur et comportant des ressources de calcul tel qu'un microprocesseur, une mémoire programmable, une mémoire morte et un interpréteur de langage, comprend, en référence avec la figure 2c précédemment introduite dans la description, au moins, outre le tableau TAB-STD des codes standard constitutifs d'un programme de type code objet intermédiaire mémorisé au niveau de l'interpréteur, un ensemble de fichiers mémorisés dans la mémoire programmable 18a par exemple.

Ainsi, l'objet portatif correspondant comprend au moins un programme de type code objet intermédiaire compacté, c'est-à-dire le fichier FCC représenté en figure 2c. Ce fichier peut être constitutif d'une application telle que mentionnée précédemment, soit d'une fonction telle qu'une fonction de chiffrement/déchiffrement de données ou analogue. Ce fichier de type code objet intermédiaire

compacté consiste bien entendu en une suite de codes d'instructions spécifiques C_i et de codes d'instructions standard correspondant aux codes d'instructions du programme de type code objet intermédiaire précité. Les codes d'instructions spécifiques C_i correspondent à des séquences d'instructions standard successives S_i précédemment mentionnées dans la description.

En outre, ainsi que représenté sur la figure 2c précitée, une table d'exécution permet la mise en correspondance biunivoque entre chaque code opératoire spécifique C_i et la séquence d'instructions standard successives S_i associée à ce dernier. L'ensemble de ces fichiers permet d'optimiser l'espace mémoire occupé dans la mémoire, notamment la mémoire programmable 18a, de l'objet portatif.

Ainsi que représenté d'ailleurs en figure 2c, la table d'exécution comprend au moins un fichier des séquences successives correspondant aux instructions spécifiques, fichier désigné par la mémoire MEM-SEQ, et un tableau, désigné par TAB-PRO des codes d'instructions spécifiques et des adresses d'implantation de ces instructions spécifiques dans le fichier des séquences successives.

L'exécution du programme de type code objet intermédiaire compacté est alors réalisée, ainsi que représenté en figure 4.

Un système de compactage d'un programme de type code objet intermédiaire permettant la mise en œuvre du procédé de compactage précédemment décrit dans la description sera maintenant donné en liaison avec les figures 6a et 6b.

D'une manière générale, le système de compactage, objet de la présente invention, sera décrit comme une combinaison de modules, ces modules pouvant être mis en

œuvre, soit de manière matérielle, soit, préférentiellement, de manière logicielle, les flux de données entre ces modules étant représentés.

Ainsi, sur la figure 6a, on a représenté le système de compactage, objet de la présente invention, lequel est réputé comprendre au moins un module A d'analyse de toutes les instructions directement exécutables, constitutives du programme de type code objet intermédiaire, noté COD-OBJ-INT. D'une manière générale, le fichier informatique support du programme de type code objet intermédiaire précité est considéré comme une chaîne d'octets, ou chaîne de caractères, et le mode opératoire du système de compactage, objet de la présente invention, sera donné dans une optique de traitement de chaîne correspondant.

A partir de la chaîne d'octets précitée, le module d'analyse A permet, par lecture du programme de type code objet COD-OBJ-INT, de discriminer et établir une liste de toutes les séquences d'instructions standard S_i contenues dans le programme précité. Sur la figure 6a, les séquences d'instructions standard S_1, S_{i-1}, S_i, S_{i+1}, ... S_p, sont ainsi notées sous forme symbolique d'une liste selon la notation symbolique des listes. On comprend ainsi que le module d'analyse A peut consister en une fenêtre glissante correspondant à un nombre n_i d'octets, cette fenêtre glissante permettant d'assurer l'analyse des séquences S_i ainsi que précédemment mentionnées en référence avec le tableau 1 de la description. La fenêtre glissante assure en fait une discrimination de chaque séquence S_i par défilement relatif de la chaîne d'octets vis-à-vis de la fenêtre précitée. A chaque occurrence de la séquence S_i considérée, un bit de comptage BC est délivré par le module d'analyse A.

Ainsi que représenté en outre en figure 6a, le système de compactage, objet de la présente invention, comprend un module C de comptage du nombre d'occurrences dans le programme de type code objet précité de chacune des séquences d'instructions directement exécutables S_i précédemment mentionnées. Le module de comptage C peut être réalisé par un module logiciel, lequel compte le nombre de bits successifs à la valeur 1 du bit de comptage BC précité. Le module de comptage C permet de mémoriser les nombres d'occurrences $N_1 \ldots N_{i-1}, N_i, N_{i+1} \ldots N_p$ de chaque séquence $S_1 \ldots S_{i-1}$ à $S_{i+1} \ldots S_p$ correspondante et suivante. Cette mémorisation peut être effectuée sous forme d'une liste.

En outre, ainsi que représenté sur la figure 6a, un module AL d'allocation à au moins une séquence d'instructions directement exécutables S_i d'un code spécifique Ci associé à cette séquence S_i est prévu pour engendrer une instruction spécifique, notée IS_i sur la figure 6a, sur critère de supériorité de la fonction d'au moins le nombre N_i d'occurrences correspondant vis-à-vis d'une valeur de référence ainsi que mentionné précédemment dans la description.

Dans le cas où la fonction d'au moins le nombre N_i est supérieure à la valeur de la fonction de la valeur de référence précitée, le module AL délivre une commande de compactage COM-COMP, lequel peut consister en un bit à la valeur 1 ou 0 correspondante.

Enfin, le système de compactage, objet de la présente invention, comprend un module de compactage proprement dit COMP, lequel reçoit, d'une part, le fichier relatif au programme de type code objet intermédiaire précité COD-OBJ-INT et la commande de comptage COM-COMP. Le module de compactage proprement dit COMP permet en fait

d'assurer le remplacement dans le programme de type code objet précité, considéré comme une chaîne d'octets, de chaque occurrence de toute séquence S_i correspondant à une instruction spécifique IS_i par le code spécifique C_i associé à cette séquence d'instructions.

En ce qui concerne le mode opératoire du module de compactage COMP proprement dit, on indique que celui-ci peut comprendre un sous-module de lecture par fenêtre glissante analogue à celui du module d'analyse, permettant de localiser la séquence d'instructions standard S_i dans la chaîne d'octets précitée. En pratique, sur localisation de la séquence d'instructions standard S_i précitée, ainsi que représenté de manière illustrative en figure 6a, le module de compactage peut comprendre un sous-module de partition à gauche et de partition à droite de la séquence S_i considérée, pour engendrer une chaîne gauche, notée LS, et une chaîne droite, notée RS. Il peut comporter ensuite, à partir du code spécifique C_i constitutif de l'instruction spécifique IS_i, un module de concaténation permettant, d'une part, la concaténation du code spécifique C_i correspondant, considéré comme une chaîne d'octets, à la chaîne gauche LS par exemple, puis concaténation de l'ensemble ainsi formé à la chaîne droite RS, ce qui permet d'assurer le remplacement de la séquence S_i par le code spécifique C_i. Le module de compactage proprement dit COMP délivre ainsi un programme de type code objet intermédiaire compacté, noté sur la figure 6a, COD-OBJ-INT-COMP. Bien entendu, le système de compactage représenté en figure 6a permet l'application du processus de compactage précédemment décrit à l'ensemble de toutes les séquences d'instructions directement exécutables S_i considérées.

En ce qui concerne le module d'allocation AL, dans un mode de réalisation non limitatif, on indique que celui-ci, ainsi que représenté en figure 6b, peut comporter un module de calcul en nombre d'octets de la longueur n_i de la séquence d'instructions S_i, ce module étant désigné par AL_1 sur la figure 6b. Il peut comporter également un module de calcul, noté AL_2, du produit de cette longueur n_i et du nombre d'occurrences N_i de cette séquence S_i d'instructions standard. Ce produit, noté P_i, est représentatif du gain de compactage pour la séquence d'instructions directement exécutables S_i considérée.

En outre, le module d'allocation AL peut comprendre un module de comparaison, noté AL_3, de ce produit P_i à une valeur de seuil, notée S, déterminée. La valeur du seuil S peut être déterminée expérimentalement. Elle peut également être établie à partir de cette valeur expérimentale pour correspondre, pour un programme de type code objet intermédiaire de longueur donnée, à un pourcentage donné de cette longueur.

Sur réponse négative au test de comparaison effectué par le module AL_3, le rang i de chaque séquence d'instructions directement exécutables S_i est incrémenté d'une unité et la nouvelle valeur de i est renvoyée au module d'analyse A, d'une part, et au module de comptage C, d'autre part.

Sur réponse positive au test de comparaison réalisé par le module AL_3, un module AL_4 permet d'établir un code spécifique C_i correspondant et, enfin, un module AL_5 permet d'assurer en correspondance biunivoque l'écriture du code spécifique C_i et de la séquence S_i considérée d'instructions directement exécutables pour constituer l'instruction spécifique IS_i.

En ce qui concerne le module AL_4, on indique que celui-ci peut être réalisé par un module logiciel de comptage permettant, à partir d'une valeur de départ, par exemple la valeur 106 précédemment mentionnée dans la description, d'allouer une valeur correspondante pour la séquence d'instructions S_i considérée. Chaque instruction spécifique IS_i peut alors être écrite sous forme d'une liste correspondante.

Des essais en temps réel de compactage de programmes ou applications contenus dans des cartes à microprocesseur commercialisées par la société BULL CP8 en France ont montré un gain de compactage supérieur à 33%, ce qui permet en fait, lors d'une application du processus de compactage à un nombre égal à trois applications pour un objet portatif mobile, de gagner sensiblement une application supplémentaire pour ce type d'objet.

Un tel gain de compactage a été obtenu dans des conditions sensiblement normales d'utilisation par l'utilisateur, alors que le ralentissement introduit par l'appel de macro-instructions, ce ralentissement étant inhérent à l'appel successif en lecture au niveau du tableau TAB-B-PRO et du fichier MEM-SEQ, n'excède pas sensiblement 10% du temps d'exécution en l'absence de macro-instructions.

REVENDICATIONS

1. Procédé de compactage d'un programme intermédiaire consistant en une suite d'instructions standard, utilisé dans un système embarqué, ce système embarqué étant doté d'une mémoire et d'un interpréteur de langage du programme intermédiaire en instructions d'un code objet directement exécutables par un microprocesseur, procédé suivant lequel :

a) on recherche dans le programme intermédiaire des séquences identiques d'instructions standard successives ;

b) on soumet les séquences identiques d'instructions successives à un test de comparaison de supériorité d'une fonction d'au moins le nombre d'occurrences de ces séquences dans ledit programme intermédiaire à une valeur de référence et, sur réponse positive audit test, pour chaque séquence identique d'instructions standard successives satisfaisant à ladite étape de test,

c) on engendre une instruction spécifique par définition d'un code opératoire spécifique et association à ce code opératoire spécifique de ladite séquence d'instructions standard successives ayant satisfait audit test ;

d) on remplace dans ledit programme intermédiaire chaque occurrence de chaque séquence d'instructions successives par ledit code opératoire spécifique qui lui est associé pour obtenir un programme intermédiaire compacté, consistant en une succession d'instructions standard et de codes opératoires spécifiques, et

e) on mémorise dans ladite mémoire une table d'exécution permettant la mise en correspondance biunivoque entre chaque code opératoire spécifique introduit et la séquence d'instructions successives associée à ce

dernier, ce qui permet d'optimiser l'espace mémoire occupé par ledit programme intermédiaire compacté par mémorisation dans ladite mémoire d'une seule occurrence desdites séquences identiques d'instructions successives.

2. Procédé selon la revendication 1, caractérisé en ce que ladite fonction est en outre fonction de la taille de chaque séquence identique d'instructions successives.

3. Procédé selon la revendication 1, caractérisé en ce que pour la mise en œuvre d'un compactage d'une pluralité de programmes intermédiaires, ledit procédé consiste en outre :

- à mémoriser la table d'exécution relative à au moins un programme intermédiaire compacté, et pour tout programme intermédiaire supplémentaire soumis à un processus de compactage ;

- à lire ladite table d'exécution mémorisée, et

- à effectuer le compactage de tout programme supplémentaire, compte tenu des instructions et codes spécifiques mémorisés dans cette table d'exécution.

4. Procédé d'exécution d'un programme intermédiaire compacté obtenu par la mise en œuvre du procédé de compactage selon la revendication 1, et consistant en une succession d'instructions standard et de codes opératoires spécifiques mémorisés dans la mémoire d'un système embarqué, caractérisé en ce qu'il consiste :

- à reconnaître dans ladite mémoire l'existence d'une table d'exécution mémorisée comportant au moins une séquence d'instructions successives associée à un code opératoire spécifique en correspondance biunivoque ;

- à appeler, par l'intermédiaire de l'interpréteur, une commande de lecture des instructions standard ou codes opératoires spécifiques successifs du programme

intermédiaire compacté et, en présence d'un code opératoire spécifique :

- appeler par instruction de lecture dans la mémoire ladite séquence d'instructions successives associée audit code opératoire spécifique et, en présence d'une instruction standard,
- appeler par instruction de lecture l'exécution de cette instruction.

5. Procédé selon la revendication 4, caractérisé en ce que lorsqu'une séquence d'instructions successives associée à un code opératoire spécifique est appelée, la valeur courante d'un compteur de programme est incrémentée dans une pile associée aux codes opératoires spécifiques, et un pointeur de programme pointe vers la première instruction de ladite séquence d'instructions spécifique, puis, sur exécution d'une instruction de fin de séquence d'instructions spécifiques, ledit compteur de programme est décrémenté, et l'exécution se poursuit à partir de l'instruction ou du code opératoire spécifique suivant.

6. Procédé selon la revendication 5, caractérisé en ce que la pile associée aux codes opératoires spécifiques et la pile associée aux instructions standard sont constituées par une pile unique.

7. Système embarqué multi-applications comprenant des ressources de calcul, une mémoire et un interpréteur de langage d'un programme intermédiaire en instructions directement exécutables par ces ressources de calcul, caractérisé en ce que ledit système embarqué multi-applications comporte au moins, outre un tableau des codes standard constitutifs dudit programme intermédiaire mémorisé au niveau dudit interpréteur :

- au moins un programme intermédiaire compacté, constitutif d'une application et consistant en une suite de codes d'instructions spécifiques et de codes d'instructions standard, lesdits codes d'instructions spécifiques correspondant à des séquences d'instructions standard successives ;

- une table d'exécution permettant la mise en correspondance biunivoque entre code opératoire spécifique et la séquence d'instructions standard successives associée à ce dernier, ledit au moins un programme intermédiaire compacté et ladite table d'exécution étant mémorisés dans ladite mémoire, ce qui permet d'optimiser l'espace mémoire occupé par ledit programme intermédiaire compacté par mémorisation dans ladite mémoire programmable d'une seule occurrence desdites séquences identiques d'instructions successives.

8. Système embarqué selon la revendication 7, caractérisé en ce que ladite table d'exécution comprend au moins :

- un fichier des séquences successives correspondant aux instructions spécifiques ;

- un tableau des codes d'instructions spécifiques et des adresses d'implantation de ces instructions spécifiques dans la table des séquences successives.

9. Système embarqué selon la revendication 8, caractérisé en ce que ledit fichier des séquences successives correspondant aux instructions spécifiques et ledit tableau des codes d'instructions spécifiques sont mémorisés en mémoire programmable dudit système embarqué.

10. Système de compactage d'un programme intermédiaire, ce programme intermédiaire consistant en une série d'instructions standard exécutables par une unité

cible, caractérisé en ce que ledit système comprend au moins :

- des moyens d'analyse de toutes les instructions standard exécutables permettant par lecture dudit programme intermédiaire de discriminer et établir une liste de toutes les séquences d'instructions standard exécutables contenues dans ce programme intermédiaire ;

- des moyens de comptage du nombre d'occurrences, dans ce programme intermédiaire, de chacune des séquences d'instructions standard exécutables membre de cette liste ;

- des moyens d'allocation à au moins une séquence d'instructions standard exécutables d'un code spécifique associé à cette séquence d'instructions standard exécutables pour engendrer une instruction spécifique ;

- des moyens de remplacement dans le programme de chaque occurrence de cette séquence d'instructions standard exécutables par le code spécifique associé à cette séquence d'instructions standard exécutables, représentatif de ladite instruction spécifique, ce qui permet d'engendrer un programme compacté, comprenant une succession d'instructions standard exécutables et d'instructions spécifiques.

11. Système selon la revendication 10, caractérisé en ce que lesdits moyens d'allocation à au moins une séquence d'instructions standard exécutables d'un code spécifique associé à cette séquence d'instructions standard exécutables pour engendrer une instruction spécifique comportent au moins :

- des moyens de calcul de la valeur d'une fonction d'au moins la longueur et du nombre d'occurrences de cette séquence d'instructions standard exécutables, ladite fonction étant représentative du gain de compactage pour cette séquence d'instructions standard exécutables ;

- des moyens de comparaison de la valeur de cette fonction à une valeur de seuil, et, sur réponse positive à ladite comparaison,

- des moyens d'écriture dans un fichier en correspondance biunivoque d'un code spécifique et de cette séquence d'instructions standard exécutables pour constituer ladite instruction spécifique.

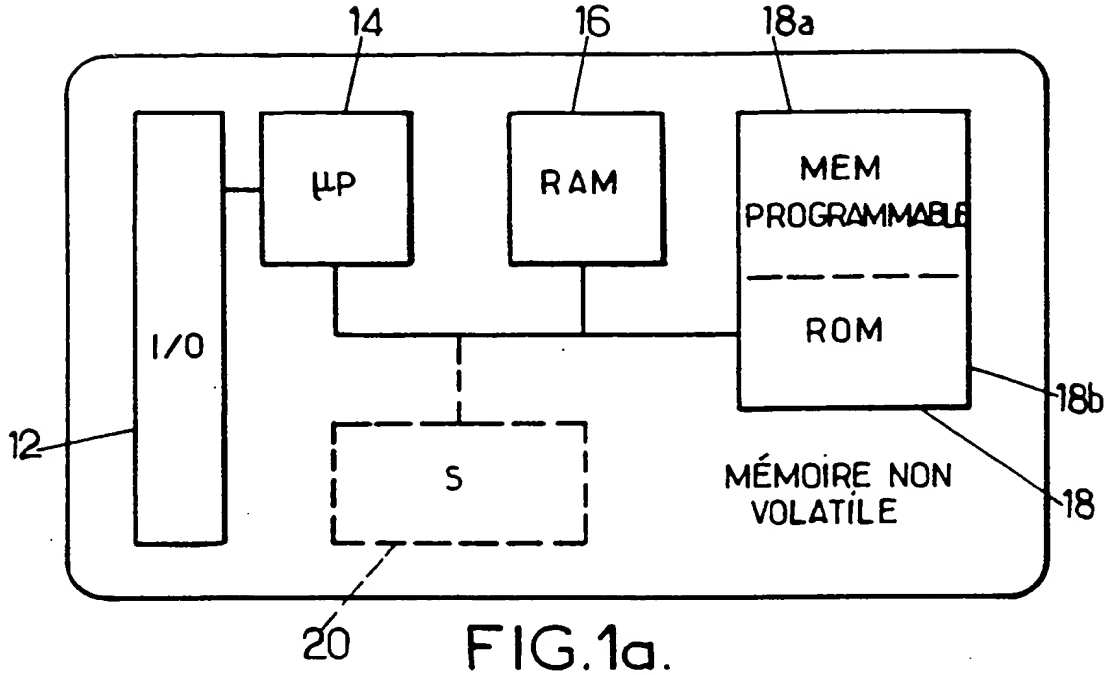


FIG.1a.
(ART ANTÉRIEUR)

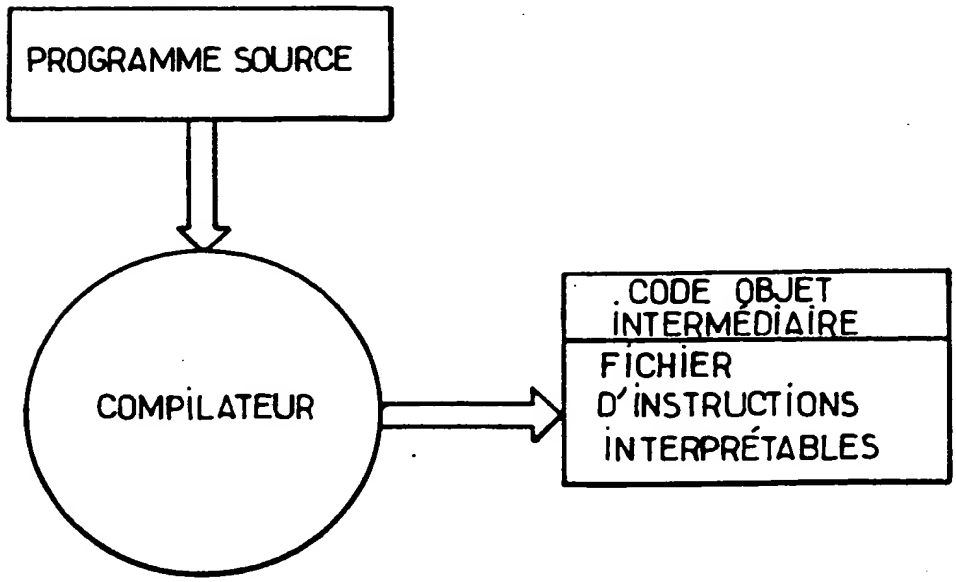


FIG.1b.
(ART ANTÉRIEUR)

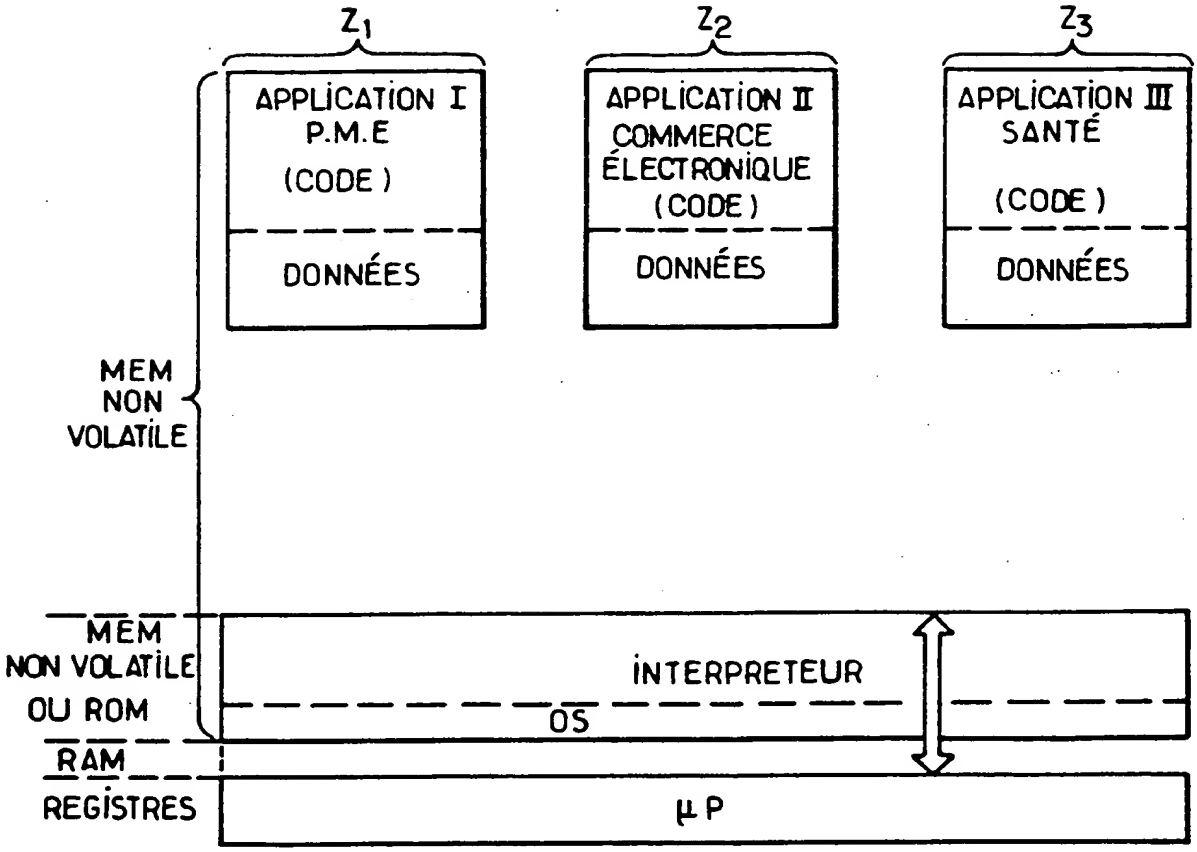


FIG.1c.
(ART ANTÉRIEUR)

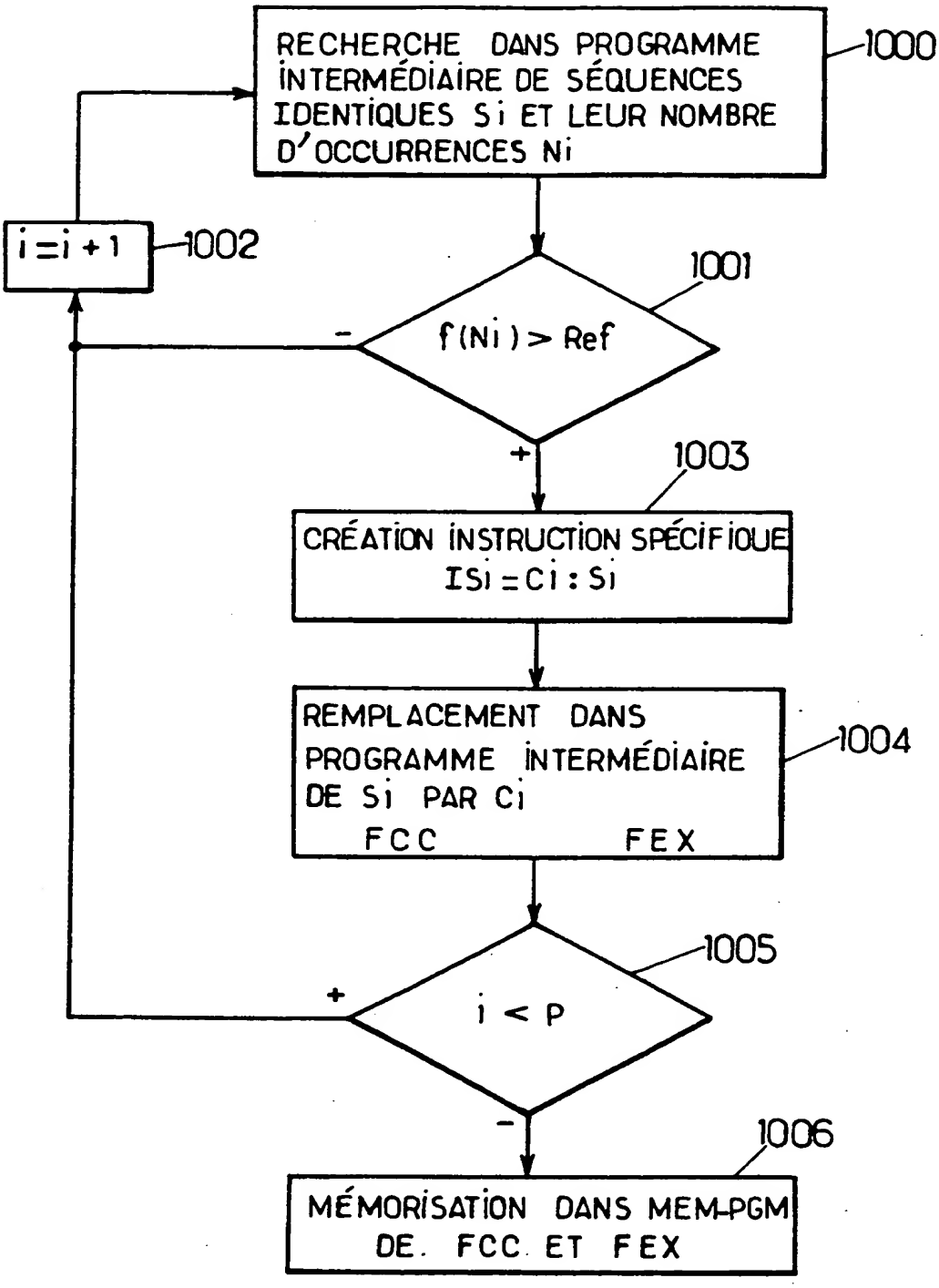

RECHERCHE DANS PROGRAMME INTERMÉDIAIRE DE SÉQUENCES IDENTIQUES Si ET LEUR NOMBRE D'OCCURRENCES Ni
1000
i=i+1
1002
1001
f(Ni) > Ref
-
+
1003
CRÉATION INSTRUCTION SPÉCIFIQUE ISi=Ci:Si
REMPLACEMENT DANS PROGRAMME INTERMÉDIAIRE DE Si PAR Ci
FCC FEX
1004
1005
i < P
+
-
1006
MÉMORISATION DANS MEM-PGM DE. FCC ET FEX

FIG.2a.

FIG.2b.

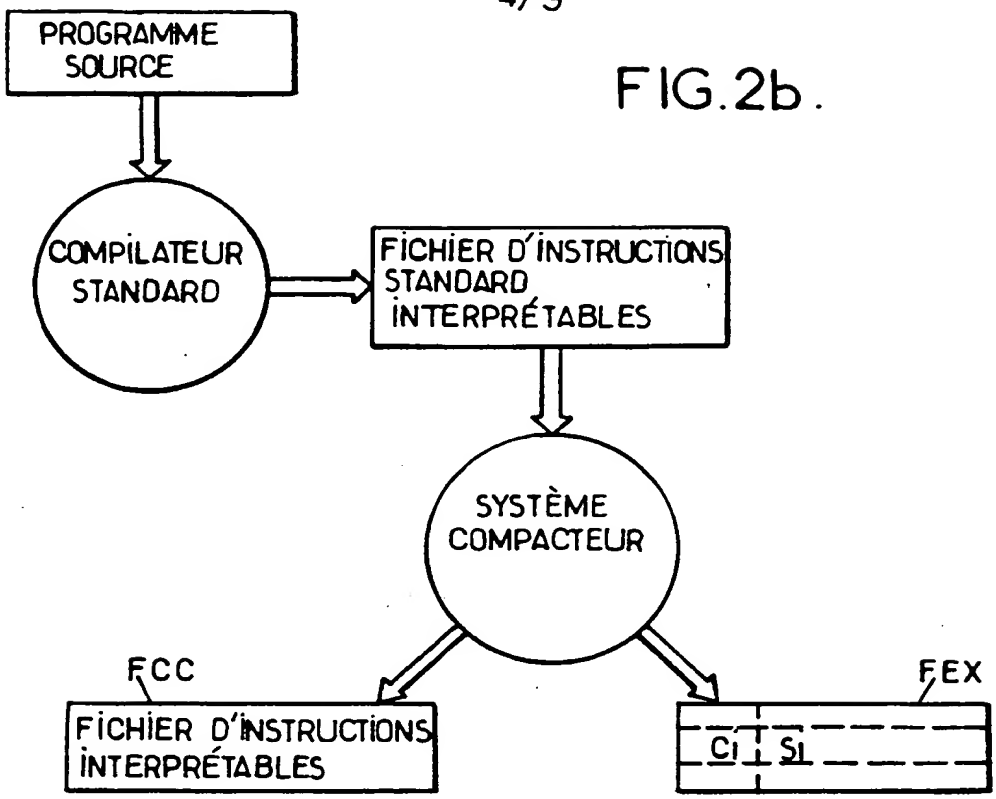
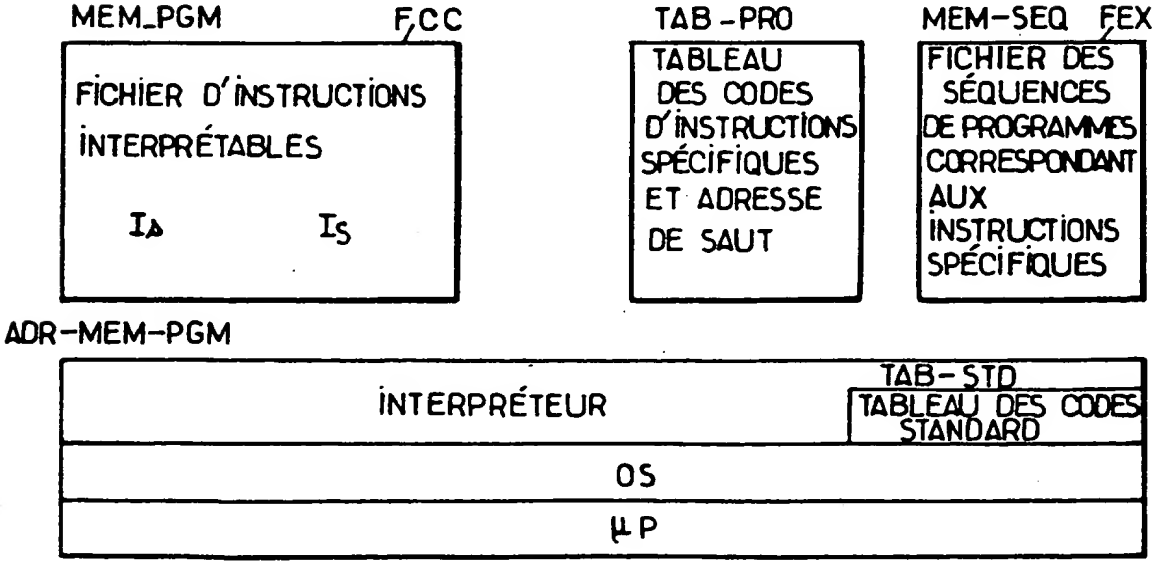

PROGRAMME SOURCE
COMPILATEUR STANDARD
FICHIER D'INSTRUCTIONS STANDARD INTERPRÉTABLES
SYSTÈME COMPACTEUR
FCC
FICHIER D'INSTRUCTIONS INTERPRÉTABLES
FEX
Ci
Si

FIG.2c.


MEM_PGM
FCC
FICHIER D'INSTRUCTIONS INTERPRÉTABLES
IA
IS
TAB-PRO
TABLEAU DES CODES D'INSTRUCTIONS SPÉCIFIQUES ET ADRESSE DE SAUT
MEM-SEQ
FEX
FICHIER DES SÉQUENCES DE PROGRAMMES CORRESPONDANT AUX INSTRUCTIONS SPÉCIFIQUES
ADR-MEM-PGM
INTERPRÉTEUR
TAB-STD
TABLEAU DES CODES STANDARD
OS
µP

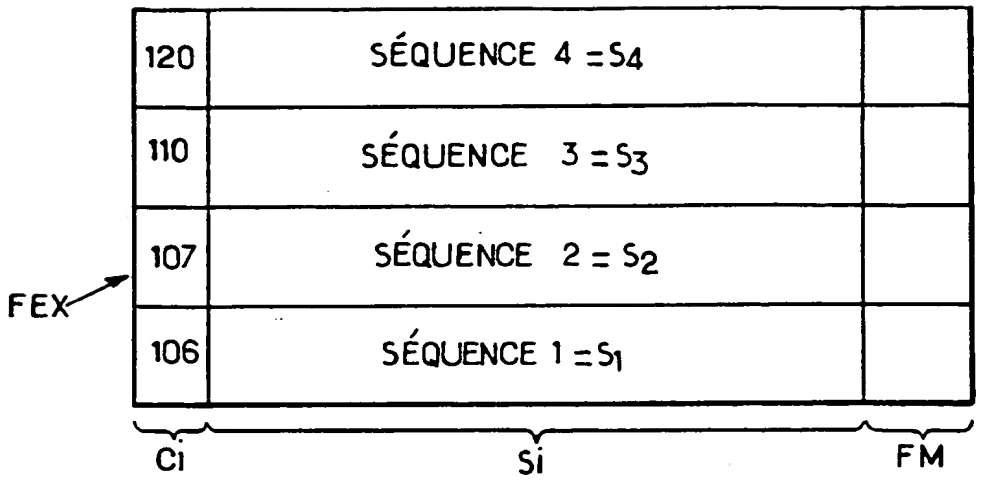

120
SÉQUENCE 4 = S4
110
SÉQUENCE 3 = S3
107
SÉQUENCE 2 = S2
106
SÉQUENCE 1 = S1
FEX
Ci
Si
FM

FIG.3a.

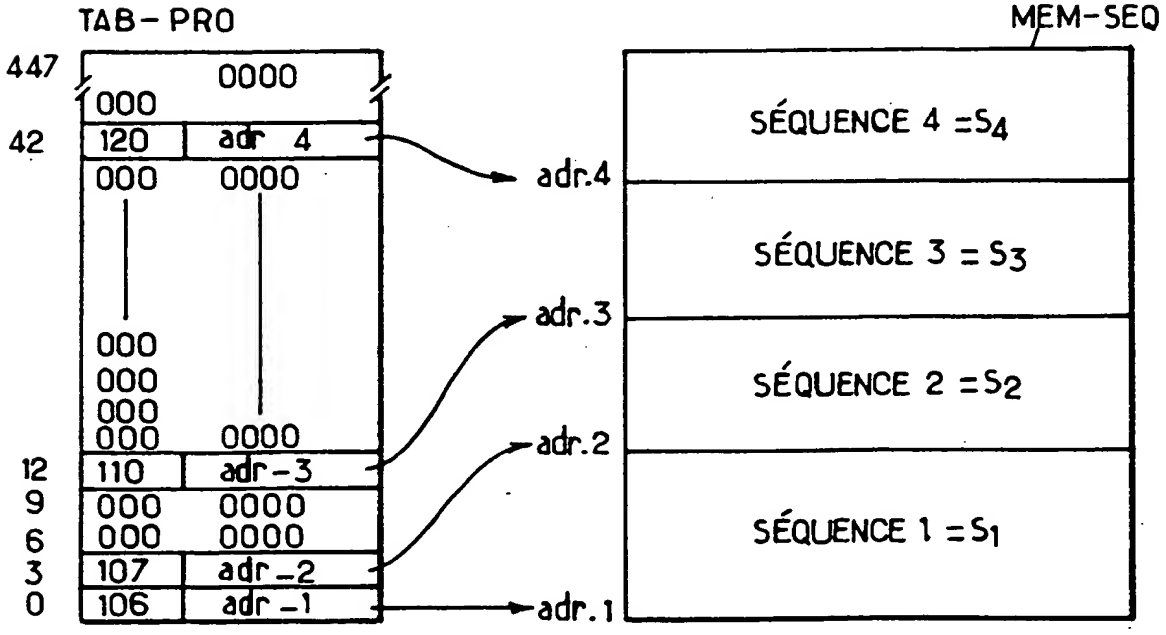
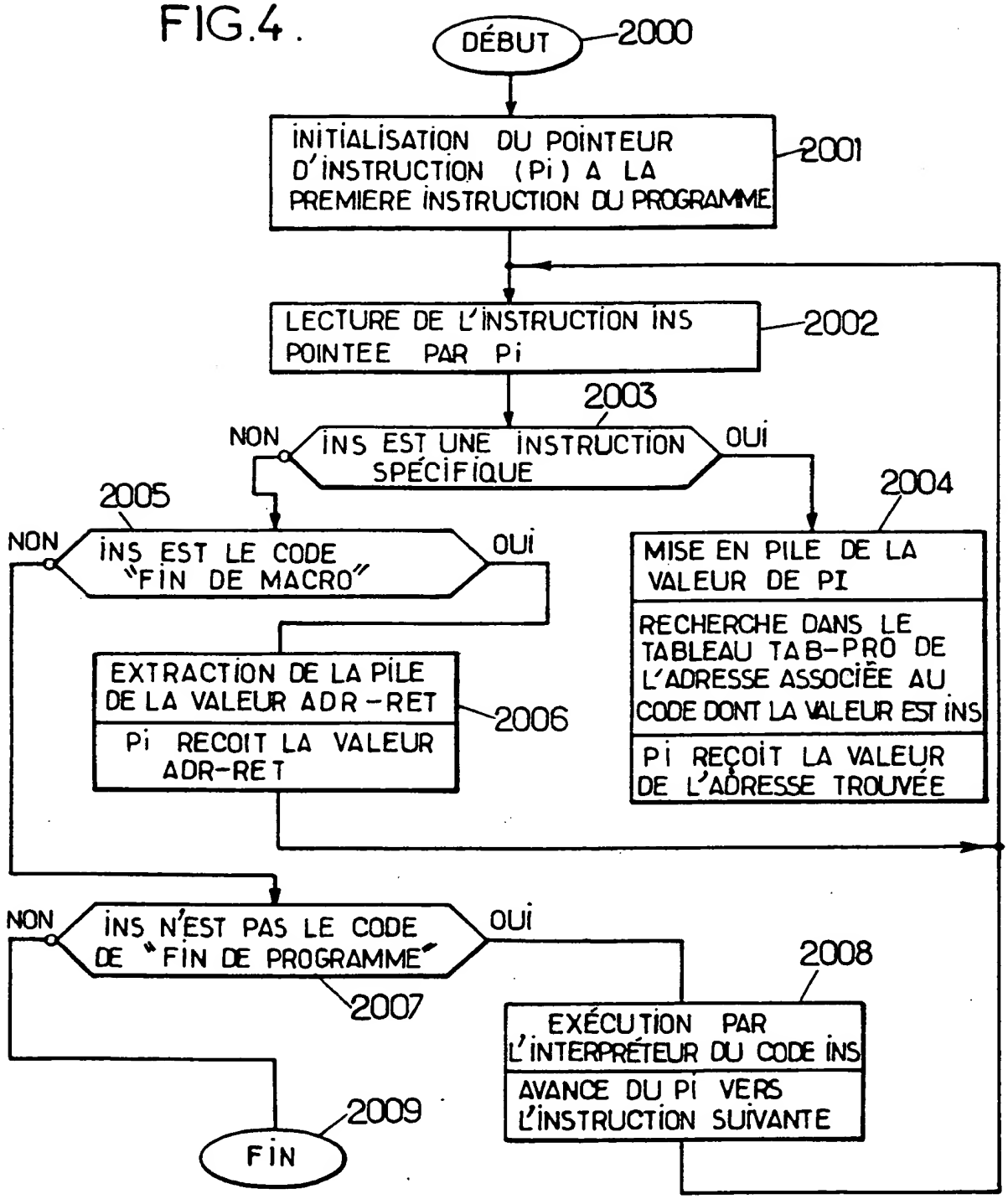

TAB-PRO
MEM-SEQ
447
0000
000
42
120
adr 4
000
0000
000
000
000
000
0000
12
110
adr-3
9
000
0000
6
000
0000
3
107
adr-2
0
106
adr-1
adr.4
adr.3
adr.2
adr.1
SÉQUENCE 4 = S4
SÉQUENCE 3 = S3
SÉQUENCE 2 = S2
SÉQUENCE 1 = S1

FIG.3b.

FIG.4.


DÉBUT
2000
INITIALISATION DU POINTEUR D'INSTRUCTION (Pi) A LA PREMIERE INSTRUCTION DU PROGRAMME
2001
LECTURE DE L'INSTRUCTION INS POINTEE PAR Pi
2002
2003
NON
INS EST UNE INSTRUCTION SPÉCIFIQUE
OUI
2004
2005
NON
INS EST LE CODE "FIN DE MACRO"
OUI
MISE EN PILE DE LA VALEUR DE PI
RECHERCHE DANS LE TABLEAU TAB-PRO DE L'ADRESSE ASSOCIÉE AU CODE DONT LA VALEUR EST INS
PI REÇOIT LA VALEUR DE L'ADRESSE TROUVÉE
EXTRACTION DE LA PILE DE LA VALEUR ADR-RET
Pi RECOIT LA VALEUR ADR-RET
2006
NON
INS N'EST PAS LE CODE DE "FIN DE PROGRAMME"
OUI
2007
2008
EXÉCUTION PAR L'INTERPRÉTEUR DU CODE INS
AVANCE DU PI VERS L'INSTRUCTION SUIVANTE
2009
FIN

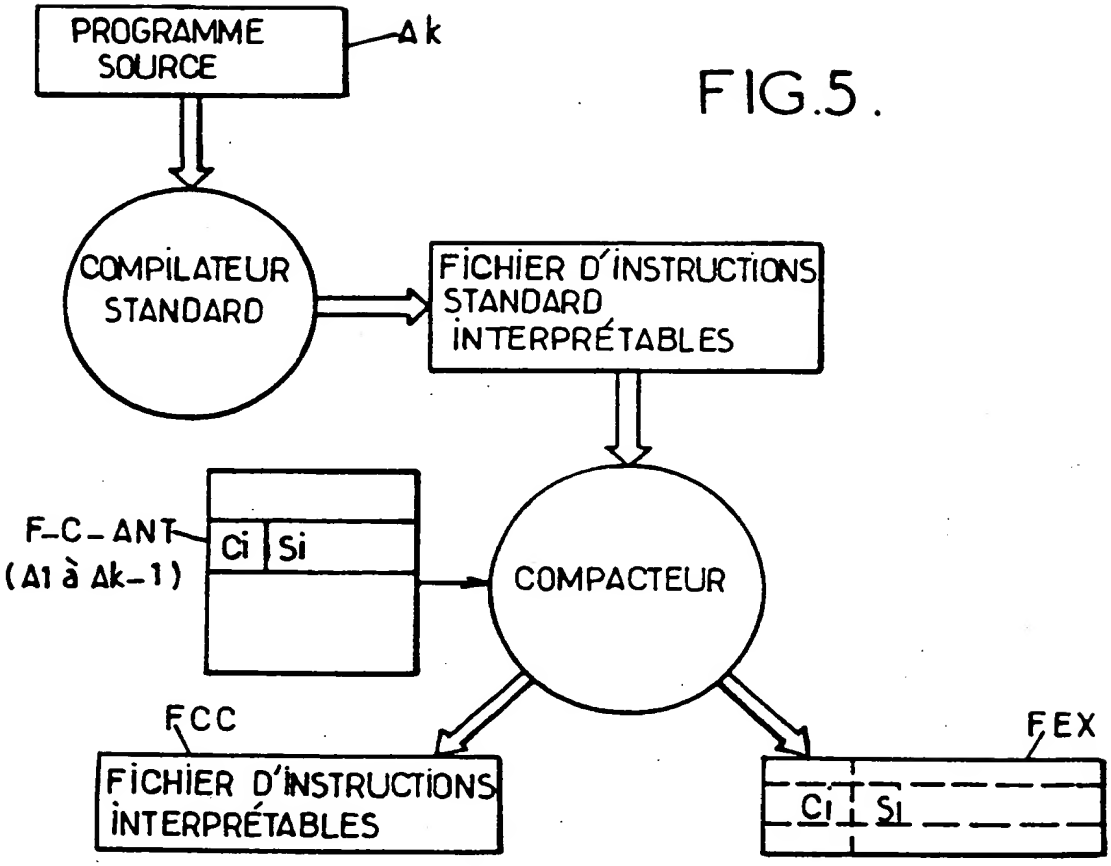

PROGRAMME SOURCE
Ak
COMPILATEUR STANDARD
FICHIER D'INSTRUCTIONS STANDARD INTERPRÉTABLES
F-C-ANT (A1 à Ak-1)
Ci
Si
COMPACTEUR
FCC
FICHIER D'INSTRUCTIONS INTERPRÉTABLES
FEX
Ci
Si

FIG.5.

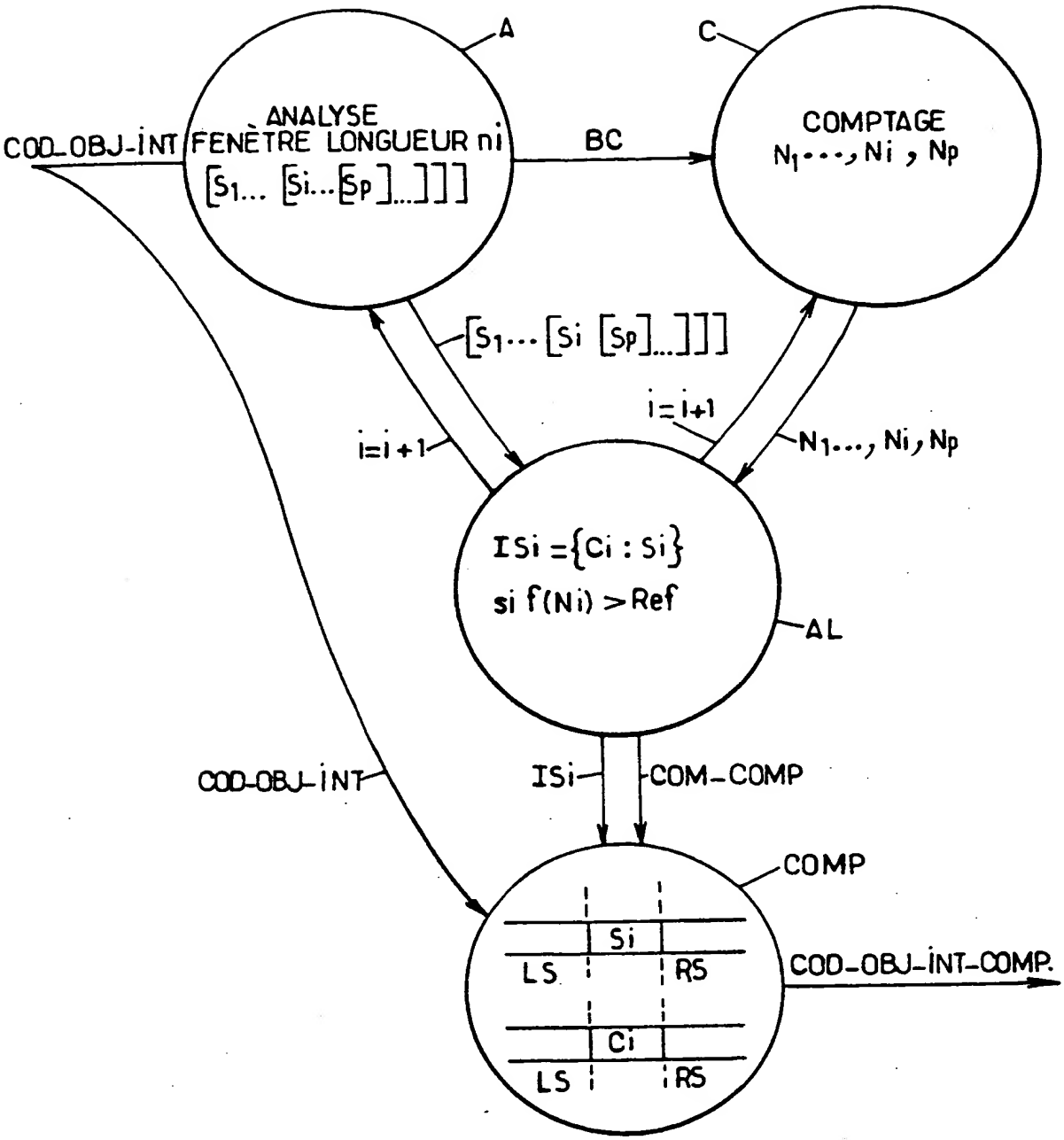
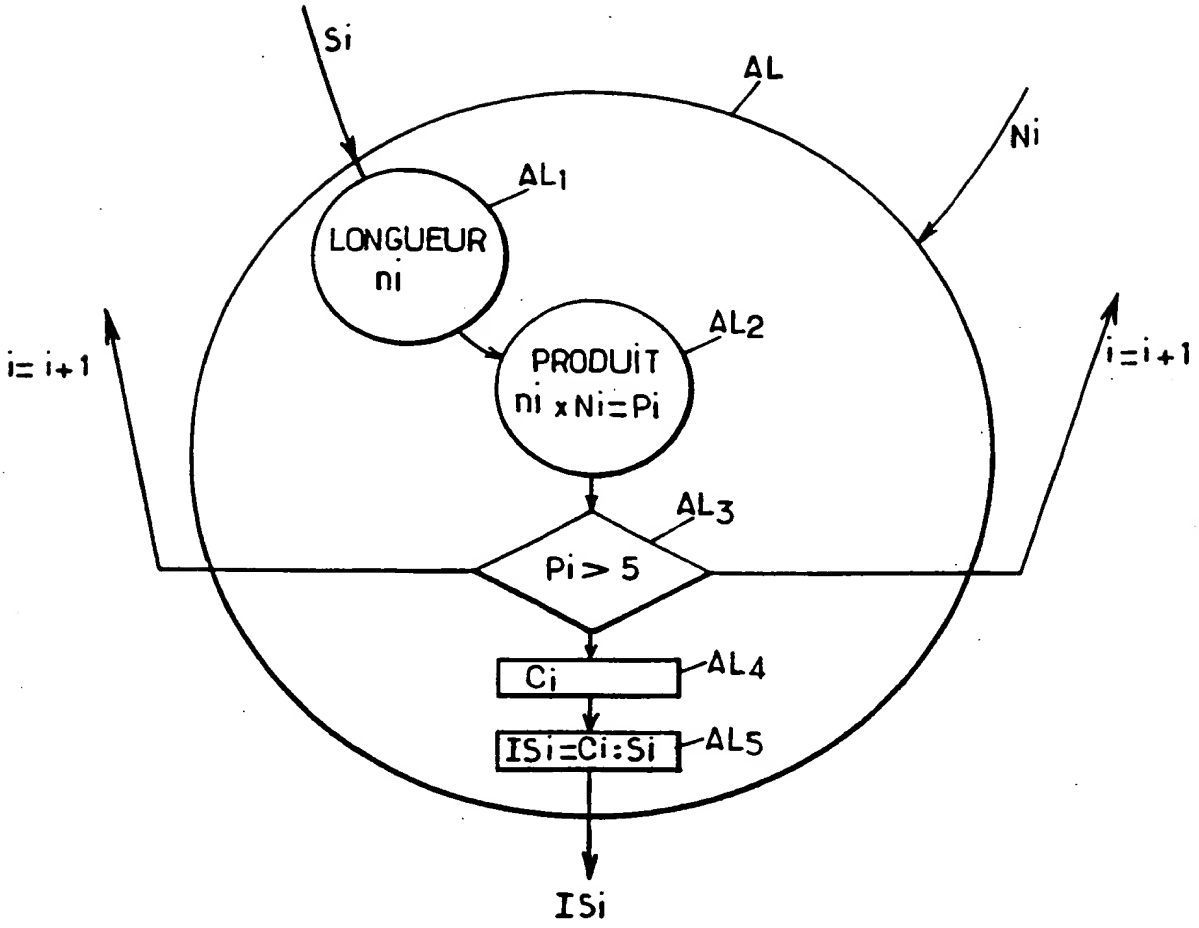

A
C
ANALYSE
FENÊTRE LONGUEUR ni
[S1... [Si...[Sp]...]]]
COMPTAGE
N1..., Ni , Np
COD-OBJ-INT
BC
[S1... [Si [Sp]...]]]
i=i+1
i=i+1
N1..., Ni, Np
ISi ={Ci : Si}
si f(Ni) > Ref
AL
COD-OBJ-INT
ISi
COM-COMP
COMP
Si
LS
RS
Ci
LS
RS
COD-OBJ-INT-COMP.

FIG.6a.

FIG.6b.


Si
AL
Ni
AL1
LONGUEUR
ni
AL2
PRODUIT
ni x Ni=Pi
i= i+1
i=i+1
AL3
Pi > 5
Ci
AL4
ISi=Ci:Si
AL5
ISi